

# New Limits for Knowledge Compilation and Applications to Exact Model Counting

**Paul Beame\***

Computer Science and Engineering  
University of Washington  
Seattle, WA 98195  
beame@cs.washington.edu

**Vincent Liew\***

Computer Science and Engineering  
University of Washington  
Seattle, WA 98195  
vliew@cs.washington.edu

August 20, 2015

## Abstract

We show new limits on the efficiency of using current techniques to make exact probabilistic inference for large classes of natural problems. In particular we show new lower bounds on knowledge compilation to SDD and DNNF forms. We give strong lower bounds on the complexity of SDD representations by relating SDD size to best-partition communication complexity. We use this relationship to prove exponential lower bounds on the SDD size for representing a large class of problems that occur naturally as queries over probabilistic databases. A consequence is that for representing unions of conjunctive queries, SDDs are not qualitatively more concise than OBDDs. We also derive simple examples for which SDDs must be exponentially less concise than FBDDs. Finally, we derive exponential lower bounds on the sizes of DNNF representations using a new quasipolynomial simulation of DNNFs by nondeterministic FBDDs.

## 1 Introduction

Weighted model counting is a fundamental problem in probabilistic inference that captures the computation of probabilities of complex predicates over independent random events (Boolean variables). Although the problem is  $\#P$ -hard in general, there are a number of practical algorithms for model counting based on DPLL algorithms and on knowledge compilation techniques. The knowledge compilation approach, though more space intensive, can be much more convenient since it builds a representation for an input predicate independent of its weights that allows the count to be evaluated easily given a particular choice of weights; that representation also can be re-used to analyze more complicated predicates. Moreover, with only a constant-factor increase in time, the methods using DPLL algorithms can be easily extended to be knowledge compilation algorithms [Huang and Darwiche, 2007]. (See [Gomes et al., 2009] for a survey.)

The representation to be used for knowledge compilation is an important key to the utility of these methods in practice; the best methods are based on restricted classes of circuits and on decision diagrams. All of the ones considered to date can be seen as natural sub-classes of the class of *Decomposable Negation Normal Form (DNNF)* formulas/circuits introduced in [Darwiche, 2001], though it is not known how to do model counting efficiently for the full class of DNNF formulas/circuits. One sub-class for which model counting is efficient given the representation is that of *d-DNNF* formulas, though there is no efficient algorithm known to recognize whether a DNNF formula is d-DNNF.

A special case of d-DNNF formulas (with a minor change of syntax) that is easy to recognize is that of *decision-DNNF* formulas. This class of representations captures all of the practical model counting algorithms discussed in [Gomes et al., 2009] including those based on DPLL algorithms. Decision-DNNFs include

---

\* Research supported by NSF grant CCF-1217099.

*Ordered Binary Decision Diagrams (OBDDs)*, which are canonical and have been highly effective representations for verification, and also *Free BDDs (FBDDs)*, which are also known as read-once branching programs. Using a quasi-polynomial simulation of decision-DNNFs by FBDDs, [Beame et al., 2013, Beame et al., 2014] showed that the best decision-DNNF representations must be exponential even for many very simple 2-DNF predicates that arise in probabilistic databases.

Recently, [Darwiche, 2011] introduced another subclass of d-DNNF formulas called *Sentential Decision Diagrams (SDDs)*. This class is strictly more general than OBDDs and (in its basic form) is similarly canonical. (OBDDs use a fixed ordering of variables, while SDDs use a fixed binary tree of variables, known as a *vtree*.) There has been substantial development and growing application of SDDs to knowledge representation problems, including a recently released SDD software package [SDD, 2014]. Indeed, SDDs hold potential to be more concise than OBDDs. [Van den Broeck and Darwiche, 2015] showed that *compressing* an SDD with a fixed vtree so that it is canonical can lead to an exponential blow-up in size, but much regarding the complexity of SDD representations has remained open.

In this paper we show the limitations both of general DNNFs and especially of SDDs. We show that the simulation of decision-DNNFs by FBDDs from [Beame et al., 2013] can be extended to yield a simulation of general DNNFs by OR-FBDDs, the nondeterministic extension of FBDDs, from which we can derive exponential lower bounds for DNNF representations of some simple functions. This latter simulation, as well as that of [Beame et al., 2013], is tight, since [Razgon, 2015a] (see also [Razgon, 2014]) shows a quasipolynomial separation between DNNF and OR-FBDD size using parameterized complexity.

For SDDs we obtain much stronger results. In particular, we relate the SDD size required to represent predicate  $f$  to the "best-case partition" communication complexity [Kushilevitz and Nisan, 1997] of  $f$ . Using this, together with reductions to the communication complexity of disjointness (set intersection), we derive the following results:

- (1) There are simple predicates given by 2-DNF formulas for which FBDD size is polynomial but for which SDD size must be exponential.
- (2) For a natural, widely-studied class of database queries known as *Unions of Conjunctive Queries (UCQ)*, the SDD size is linear iff the OBDD size is linear and is exponential otherwise (which corresponds to a query that contains an *inversion* [Jha and Suciu, 2013]).
- (3) Similar lower bounds apply to the dual of UCQ, which consists of universal, positive queries.

To prove our SDD results, we show that for any predicate  $f$  given by an SDD of size  $S$ , using its associated vtree we can partition the variables of  $f$  between two players, Alice and Bob, in a nearly balanced way so that they only need to send  $\log^2 S$  bits of communication to compute  $f$ . The characterization goes through an intermediate step involving *unambiguous* communication protocols and a clever deterministic simulation of such protocols from [Yannakakis, 1991].

**Related work:** The quasi-polynomial simulation of DNNFs by OR-FBDDs that we give was also shown independently in [Razgon, 2015b]. Beyond the lower bounds for decision-DNNFs in [Beame et al., 2013, Beame et al., 2014] which give related analyses for decision-DNNFs, the work of [Pipatsrisawat and Darwiche, 2010] on structured DNNFs is particularly relevant to this paper<sup>1</sup>. [Pipatsrisawat and Darwiche, 2010] show how sizes of what they term (*deterministic*) **X**-*decompositions* can yield lower bounds on the sizes of *structured (deterministic)* DNNFs, which include SDDs as a special case. [Pipatsrisawat, 2010] contains the full details of how this can be applied to prove lower bounds for specific predicates. These bounds are actually equivalent to lower bounds exponential in the best-partition non-deterministic (respectively, unambiguous) communication complexity of the given predicates. Our paper derives this lower bound for SDDs directly but, more importantly, provides the connection to best-partition *deterministic* communication complexity, which allows us to have a much wider range of application; this strengthening is necessary for our applications. Finally, we note that [Razgon, 2014] showed that SDDs can be powerful by finding examples where OBDDs using any order are quasipolynomially less concise than SDDs.

---

<sup>1</sup>We thank the conference reviewers for bringing this work to our attention.

**Roadmap:** We give the background and some formal definitions including some generalization required for this work in Section 2. We prove our characterization of SDDs in terms of best-partition communication complexity in Section 3 and derive the resulting bounds for SDDs for natural predicates in Section 4. We describe the simulation of DNNFs by OR-FBDDs, and its consequences, in Section 5.

## 2 Background and Definitions

We first give some basic definitions of DNNFs and decision diagrams.

**Definition 2.1.** A Negation Normal Form (NNF) circuit is a Boolean circuit with  $\neg$  gates, which may only be applied to inputs, and  $\vee$  and  $\wedge$  gates. Further, it is Decomposable (DNNF) iff the children of each  $\wedge$  gate are reachable from disjoint sets of input variables. (Following convention, we call this circuit a “DNNF formula”, though it is not a Boolean formula in the usual sense of circuit complexity.) A DNNF formula is deterministic (d-DNNF) iff the functions computed at the children of each  $\vee$  gate are not simultaneously satisfiable.

**Definition 2.2.** A Free Binary Decision Diagram (FBDD) is a directed acyclic graph with a single source (the root) and two specified sink nodes, one labeled 0 and the other 1. Every non-sink node is labeled by a Boolean variable and has two out-edges, one labeled 0 and the other 1. No path from the root to either sink is labeled by the same variable more than once. It is an OBDD if the order of variable labels is the same on every path. The Boolean function computed by an FBDD is 1 on input  $\mathbf{a}$  iff there is a path from the root to the sink labeled 1 so that for every node label  $X_i$  on the path,  $\mathbf{a}_i$  is the label of the out-edge taken by the path. An OR-FBDD is an FBDD augmented with additional nodes of arbitrary fan-out labeled  $\vee$ . The function value for the OR-FBDD follows the same definition as for FBDDs; the  $\vee$ -nodes simply make more than one path possible for a given input. (See [Wegener, 2000].)

We now define sentential decision diagrams as well as a small generalization that we will find useful.

**Definition 2.3.** For a set  $\mathbf{X}$ , let  $\top : \{0,1\}^{\mathbf{X}} \rightarrow \{0,1\}$  and  $\perp : \{0,1\}^{\mathbf{X}} \rightarrow \{0,1\}$  denote the constant 1 function and constant 0 function, respectively.

**Definition 2.4.** We say that a set of Boolean functions  $\{p_1, p_2, \dots, p_\ell\}$ , where each  $p_i$  has domain  $\{0,1\}^{\mathbf{X}}$ , is disjoint if for each  $i \neq j$ ,  $p_i \wedge p_j = \perp$ . We call  $\{p_1, p_2, \dots, p_\ell\}$  a partition if it is disjoint and  $\bigvee_{i=1}^{\ell} p_i = \top$ .

**Definition 2.5.** A vtree for variables  $\mathbf{X}$  is a full binary tree whose leaves are in one-to-one correspondence with the variables in  $\mathbf{X}$ .

We define *Sentential Decision Diagrams* (SDDs) together with the Boolean functions they represent and use  $\langle \cdot \rangle$  to denote the mapping from SDDs into Boolean functions. (This notation is extended to sets of SDDs yielding sets of Boolean functions.) At the same time, we also define a directed acyclic graph (DAG) representation of the SDD.

**Definition 2.6.**  $\alpha$  is an SDD that respects vtree  $\mathbf{v}$  rooted at  $v$  iff:

- $\alpha = \top$  or  $\alpha = \perp$ .  
Semantics:  $\langle \top \rangle = \top$  and  $\langle \perp \rangle = \perp$ .  
 $G(\alpha)$  consists of a single leaf node labeled with  $\langle \alpha \rangle$ .
- $\alpha = X$  or  $\alpha = \neg X$  and  $v$  is a leaf with variable  $X$ .  
Semantics:  $\langle X \rangle = X$  and  $\langle \neg X \rangle = \neg X$   
 $G(\alpha)$  consists of a single leaf node labeled with  $\langle \alpha \rangle$ .
- $\alpha = \{(p_1, s_1), \dots, (p_\ell, s_\ell)\}$ ,  $v$  is an internal vertex with children  $v_L$  and  $v_R$ ,  $p_1, \dots, p_\ell$  are SDDs that respect the subtree rooted at  $v_L$ ,  $s_1, \dots, s_\ell$  are SDDs that respect the subtree rooted at  $v_R$ , and  $\langle p_1 \rangle, \dots, \langle p_\ell \rangle$  is a partition.

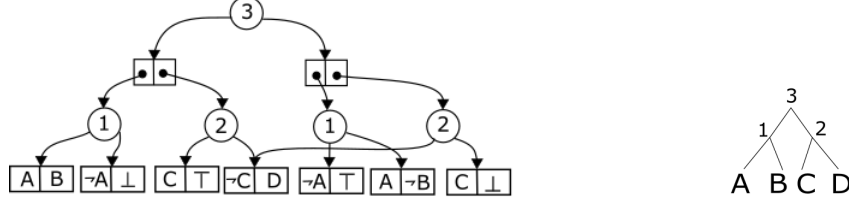


Figure 1: An SDD with its associated vtree that computes the formula  $(A \wedge B \wedge C) \vee (\neg C \wedge D)$

*Semantics:*  $\langle \alpha \rangle = \bigvee_{i=1}^n (\langle p_i \rangle \wedge \langle s_i \rangle)$

$G(\alpha)$  has a circle node for  $\alpha$  labeled  $v$  with  $\ell$  child box nodes labeled by the pairs  $(p_i, s_i)$ . A box node labeled  $(p_i, s_i)$  has a left child that is the root of  $G(p_i)$  and a right child that is the root of  $G(s_i)$ . The rest of  $G(\alpha)$  is the (non-disjoint) union of graphs  $G(p_1), \dots, G(p_\ell)$  and  $G(s_1), \dots, G(s_\ell)$  with common sub-DAGs merged. (See Figure 1.)

Each circle node  $\alpha'$  in  $G(\alpha)$  itself represents an SDD that respects a subtree of  $\mathbf{v}$  rooted at some vertex  $v'$  of  $\mathbf{v}$ ; We say that  $\alpha'$  is in  $\alpha$  and use  $\text{Sdds}(v', \alpha)$  to denote the collection of  $\alpha'$  in  $\alpha$  that respect the subtree rooted at  $v'$ . The size of an SDD  $\alpha$  is the number of nodes in  $G(\alpha)$ .

Circle nodes in  $G(\alpha)$  may be interpreted as OR gates and paired box nodes may be interpreted as AND gates. In the rest of this paper, we will view SDDs as a class of Boolean circuit. The vtree property and partition property of SDDs together ensure that this resulting circuit is a d-DNNF.

We define a small generalization of vtrees which will be useful for describing SDDs with respect to a partial assignment of variables.

**Definition 2.7.** A pruned vtree for variables  $\mathbf{X}$  is a full binary tree whose leaves are either marked stub or by a variable in  $\mathbf{X}$ , and whose leaves marked by variables are in one-to-one correspondence with the variables in  $\mathbf{X}$ .

We generalize SDDs so that they can respect pruned vtrees. The definition is almost identical to that for regular SDDs so we only point out the differences.

**Definition 2.8.** The definition of a pruned SDD  $\alpha$  respecting a pruned vtree  $\mathbf{v}$ , its semantics, and its graph  $G(\alpha)$ , are identical to those of an SDD except that

- if the root vertex  $v$  of  $\mathbf{v}$  is a stub then  $\langle \alpha \rangle$  must be  $\perp$  or  $\top$ , and
- if the root vertex  $v$  of  $\mathbf{v}$  is internal then we only require that  $\langle p_1 \rangle, \dots, \langle p_\ell \rangle$  are disjoint but not necessarily that they form a partition.

We now sketch a very brief overview of the communication complexity we will need. Many more details may be found in [Kushilevitz and Nisan, 1997]. Given a Boolean function  $f$  on  $\{0, 1\}^{\mathbf{X}} \times \{0, 1\}^{\mathbf{Y}}$ , one can define two-party protocols in which two players, Alice, who receives  $x \in \{0, 1\}^{\mathbf{X}}$  and Bob, who receives  $y \in \{0, 1\}^{\mathbf{Y}}$  exchange a sequence of messages  $m_1, \dots, m_C = f(x, y) \in \{0, 1\}$  to compute  $f$ . (After each bit, the player to send the next bit must be determined from previous messages.) The (*deterministic*) communication complexity of  $f$ ,  $CC(f(\mathbf{X}, \mathbf{Y}))$ , is the minimum value  $C$  over all protocols computing  $f$  such that all message sequences are of length at most  $C$ . The *one-way deterministic communication complexity* of  $f$ ,  $CC_{\mathbf{X} \rightarrow \mathbf{Y}}(f(\mathbf{X}, \mathbf{Y}))$  is the minimum value of  $C$  over all protocols where Alice may send messages to Bob, but Bob cannot send messages to Alice.

For nondeterministic protocols, Alice simply guesses a string based on her input  $x$  and sends the resulting message  $m$  to Bob, who uses  $m$  together with  $y$  to verify whether or not  $f(x, y) = 1$ . The communication complexity in this case is the minimum  $|m|$  over all protocols. Such a protocol is *unambiguous* iff for each  $(x, y)$  pair such that  $f(x, y) = 1$  there is precisely one message  $m$  that will cause Bob to output 1. A set of the form  $A \times B$  for  $A \subseteq \{0, 1\}^{\mathbf{X}}$ ,  $B \subseteq \{0, 1\}^{\mathbf{Y}}$  is called a *rectangle*. The minimum of  $|m|$  over all unambiguous

protocols is the *unambiguous communication complexity* of  $f$ ; it is known to be the logarithm base 2 of the minimum number of rectangles into which one can partition the set of inputs on which  $f$  is 1.

A canonical hard problem for communication complexity is the two-party disjointness (set intersection) problem,  $\bigvee_{i=1}^n x_i \wedge y_i$  where  $x$  and  $y$  are indicator vectors of sets in  $[n]$ . It has deterministic communication complexity  $n + 1$  (and requires  $\Omega(n)$  bits be sent even with randomness, but that is beyond what we need). We will need a variant of the “best partition” version of communication complexity in which the protocol includes a choice of the best split of input indices  $\mathbf{X}$  and  $\mathbf{Y}$  between Alice and Bob.

A typical method for proving lower bounds on OBDD size for a Boolean function  $f$  begins by observing that a size  $s$  OBDD may be simulated by a log  $s$ -bit one-way communication protocol where Alice holds the first half of the variables read by the OBDD and Bob holds the second half. In this protocol, Alice starts at the root of the OBDD and follows the (unique) OBDD path determined by her half of the input until she reaches a node  $v$  querying a variable held by Bob. She then sends the identity of the node  $v$  to Bob, who can finish the computation starting from  $v$ . Thus, if we show that  $f$  has one-way communication complexity  $CC_{\mathbf{X} \rightarrow \mathbf{Y}}(f(\mathbf{X}, \mathbf{Y}))$  at least  $C$  in the best split  $\{\mathbf{X}, \mathbf{Y}\}$  of its input variables, then any OBDD computing  $f$  must have at least  $2^C$  nodes.

Our lower bound for SDDs uses related ideas but in a more sophisticated way, and instead of providing a one-way deterministic protocol, we give an unambiguous protocol that simulates the SDD computation. In particular, the conversion to deterministic protocols requires two-way communication.

### 3 SDDs and Best-Partition Communication Complexity

In this section, we show how we can use any small SDD representing a function  $f$  to build an efficient communication protocol for  $f$  given an approximately balanced partition of input variables that is determined by its associated vtree. As a consequence, any function requiring large communication complexity under all such partitions requires large SDDs. To begin this analysis, we consider how an SDD simplifies under a partial assignment to its input variables.

#### 3.1 Pruning SDDs Using Restrictions

**Definition 3.1.** Suppose that  $\mathbf{v}$  is a pruned vtree for a set of variables  $\mathbf{X}$ , and that  $v$  is a vertex in  $\mathbf{v}$ . Let  $\text{Vars}(v)$  denote the set of variables that are descendants of  $v$  in  $\mathbf{v}$  and  $\text{Shell}(v) = \mathbf{X} \setminus \text{Vars}(v)$ . Also let  $\text{Parent}(v)$  denote the (unique) vertex in  $\mathbf{v}$  that has  $v$  as a child.

We define a construction to capture what happens to an SDD under a partial assignment of its variables.

**Definition 3.2.** Let  $\alpha$  be an SDD that respects  $\mathbf{v}$ , a vtree for the variables  $\mathbf{X}$ , and suppose that  $\alpha$  computes the function  $f$ . Let  $\mathbf{B} \subseteq \mathbf{X}$  and  $\mathbf{A} = \mathbf{X} \setminus \mathbf{B}$  and let  $\rho : \mathbf{A} \rightarrow \{0, 1\}$  be an assignment to the variables in  $\mathbf{A}$ . Let  $\alpha|_\rho$  be Boolean circuit remaining after plugging the partial assignment  $\rho$  into the SDD  $\alpha$  and making the following simplifications:

1. If a gate computes a constant  $c \in \{\top, \perp\}$  under the partial assignment  $\rho$ , we can replace that gate and its outgoing edges with  $c$ .
2. Remove any children of OR-gates that compute  $\perp$ .
3. Remove any nodes disconnected from the root.

For each vtree vertex  $v \in \mathbf{v}$  that was not removed in this process, we denote its counterpart in the pruned vtree  $\mathbf{v}|_{\mathbf{A}}$  by  $v|_{\mathbf{A}}$ .

Construct the pruned vtree  $\mathbf{v}|_{\mathbf{A}}$  from  $\mathbf{v}$  as follows: for each vertex  $v$ , if  $\text{Vars}(v) \subseteq \mathbf{A}$  and  $\text{Vars}(\text{Parent}(v)) \not\subseteq \mathbf{A}$ , replace  $v$  and its subtree by a stub. We say that we have pruned the subtree rooted at  $v$ . (See Figure 2 for an example of an SDD and its vtree both before and after pruning.)

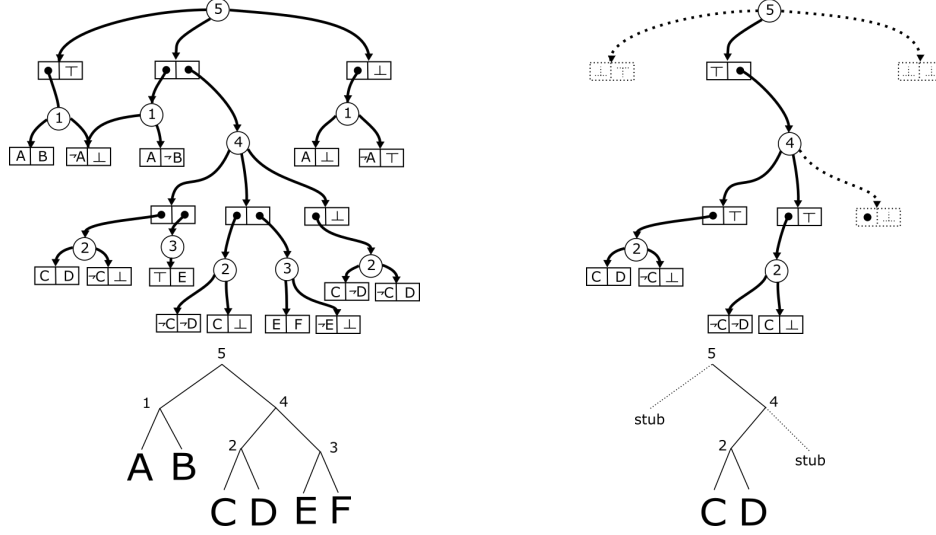


Figure 2: An SDD and its vtree, as well as the pruned pair after setting  $B$  to 0 and  $A, E, F$  to 1.

For  $\mathbf{A} \subseteq \mathbf{X}$ , we call  $\{\mathbf{A}, \mathbf{X} \setminus \mathbf{A}\}$  a shell partition for  $\mathbf{X}$  if there is a vtree vertex  $v \in \mathbf{v}$  such that  $\text{Shell}(v) = \mathbf{A}$ . We call  $\mathbf{A}$  the shell. If, for a restriction  $\rho : \mathbf{A} \rightarrow \{0, 1\}$ , there exists a vtree vertex  $v \in \mathbf{v}$  such that  $\text{Shell}(v) = \mathbf{A}$ , we call  $\rho$  a shell restriction.

**Proposition 3.3.** Let  $\alpha$  be an SDD that respects  $\mathbf{v}$ , a vtree for the variables  $\mathbf{X}$ , and suppose that  $\alpha$  computes the function  $f$ . Let  $\mathbf{A} \subseteq \mathbf{X}$  and  $\rho : \mathbf{A} \rightarrow \{0, 1\}$  be a partial assignment of the variables in  $\mathbf{A}$ . The pruned SDD  $\alpha|_\rho$  has the following properties:

- (a)  $\langle \alpha|_\rho \rangle = f|_\rho$ .
- (b)  $\alpha|_\rho$  is a pruned SDD respecting  $\mathbf{v}|_{\mathbf{A}}$ .
- (c)  $G(\alpha|_\rho)$  is a subgraph of  $G(\alpha)$ .

*Proof.* (a): An SDD may be equivalently described as a Boolean circuit of alternating OR and AND gates. For any Boolean circuit in the variables  $\mathbf{X}$  that computes  $f$ , plugging in the values for the restriction  $\rho$  yields a circuit computing  $f|_\rho$ . Furthermore, the simplification steps do not change the function computed.

(b): For each  $v$  such that  $\text{Vars}(v) \subseteq \mathbf{A}$  and  $\text{Vars}(\text{Parent}(v)) \not\subseteq \mathbf{A}$ , we have replaced the subtree rooted at  $v$  by a stub and replaced the SDDs in  $\alpha$  respecting  $\mathbf{v}$  by either  $\top$  or  $\perp$ . Thus  $\alpha|_\rho$  respects  $\mathbf{v}|_{\mathbf{A}}$ .

We now check that  $\alpha|_\rho$  is a pruned SDD. In particular we need to ensure that for each SDD  $\alpha' = \{(p_1, s_1), \dots, (p_\ell, s_\ell)\}$  in  $\alpha$ , the corresponding pruned SDDs that remain from  $p_1, \dots, p_\ell$  in its pruned counterpart  $\alpha'|_\rho$  represent a collection of disjoint functions. From the first part of this proposition, these are  $\langle p_{i_1} \rangle|_\rho, \dots, \langle p_{i_k} \rangle|_\rho$  for some  $k \leq n$ , where we have only included those SDDs that are consistent under  $\rho$ . Since the original set of SDDs was a partition and thus disjoint, this set of restricted (pruned) SDDs is also disjoint.

(c): The process in Definition 3.2 only removes nodes from  $G(\alpha)$  to construct  $G(\alpha|_\rho)$ . Further, it does not change the label of any SDD that was not removed.  $\square$

### 3.2 Unambiguous Communication Protocol for SDDs

The way that we will partition the input variables to an SDD between the parties Alice and Bob in the communication protocol will respect the structure of its associated vtree. The restrictions will correspond to assignments that reflect Alice's knowledge of the input and will similarly respect that structure.

Notice that a vtree cut along an edge  $(u, v)$  (where  $u$  is the parent of  $v$ ) induces a shell partition for  $\mathbf{X}$  consisting of the set  $\mathbf{B} = \text{Vars}(v)$ , and the shell  $\mathbf{A} = \mathbf{X} \setminus \mathbf{B}$ .

**Proposition 3.4.** *Let  $\alpha$  be an SDD of size  $s$  computing a function  $f : \{0, 1\}^{\mathbf{X}} \rightarrow \{0, 1\}$  that respects a vtree  $\mathbf{v}$ . Suppose that  $\{\mathbf{A}, \mathbf{B}\}$  is a shell partition for  $\mathbf{X}$  and that  $\mathbf{A}$  is its shell. Let  $b$  be the vertex in  $\mathbf{v}$  for which  $\text{Vars}(b) = \mathbf{B}$  and  $\text{Vars}(\text{Parent}(b)) \not\subseteq \mathbf{B}$ .*

*For any shell restriction  $\rho : \mathbf{A} \rightarrow \{0, 1\}$ , the set  $\langle \text{Sdds}_{\alpha|_{\rho}}(b|_{\mathbf{A}}) \rangle$  is a disjoint collection of functions.*

*Proof.* For non-shell restrictions  $\rho'$ , the collection of functions  $\langle \text{Sdds}_{\alpha|_{\rho'}}(v) \rangle$  for a vtree node  $v$  is not disjoint; we need to use the specific properties of  $\mathbf{A}$  and  $b$ . Since  $\rho$  was a shell restriction, the pruned vtree  $\mathbf{v}|_{\mathbf{A}}$  takes the form of a path  $v_1|_{\mathbf{A}}, \dots, v_k|_{\mathbf{A}}$  of internal vertices, where  $v_1$  is the root of  $\mathbf{v}$ , and  $v_k|_{\mathbf{A}} = b|_{\mathbf{A}}$ , with the other child of each of  $v_1|_{\mathbf{A}}, \dots, v_{k-1}|_{\mathbf{A}}$  being a stub, together with a vtree for the variables  $\mathbf{B}$  rooted at  $b$ . We will show that if  $\langle \text{Sdds}_{\alpha|_{\rho}}(v_i|_{\mathbf{A}}) \rangle$  is disjoint then so is  $\langle \text{Sdds}_{\alpha|_{\rho}}(v_{i+1}|_{\mathbf{A}}) \rangle$ . This will prove the proposition since  $\langle \text{Sdds}_{\alpha|_{\rho}}(v_1|_{\mathbf{A}}) \rangle$  only contains the function  $\langle \alpha|_{\rho} \rangle$  and is therefore trivially disjoint.

We will use the fact that every pruned-SDD from  $\text{Sdds}_{\alpha|_{\rho}}(v_{i+1}|_{\mathbf{A}})$  is contained in some SDD from  $\text{Sdds}_{\alpha|_{\rho}}(v_i|_{\mathbf{A}})$ . We have two cases to check:  $v_{i+1}|_{\mathbf{A}}$  is either a left child or a right child of  $v_i|_{\mathbf{A}}$ .

If  $v_{i+1}|_{\mathbf{A}}$  was a right child then each pruned-SDD  $\eta|_{\rho}$  contained in  $\text{Sdds}_{\alpha|_{\rho}}(v_i|_{\mathbf{A}})$  takes the form  $\eta|_{\rho} = \{(\top, s|_{\rho})\}$ . Then  $\langle \text{Sdds}_{\alpha|_{\rho}}(v_{i+1}|_{\mathbf{A}}) \rangle = \langle \text{Sdds}_{\alpha|_{\rho}}(v_i|_{\mathbf{A}}) \rangle$  and is therefore disjoint by assumption.

Otherwise suppose that  $v_{i+1}|_{\mathbf{A}}$  is the left child of  $v_i|_{\mathbf{A}}$ . Let  $\eta|_{\rho} \in \text{Sdds}_{\alpha|_{\rho}}(v_i|_{\mathbf{A}})$ . Let  $\eta|_{\rho} = \{(\eta_1|_{\rho}, \top), \dots, (\eta_k|_{\rho}, \top)\}$  where  $\bigvee_{i=1}^k \langle \eta_i|_{\rho} \rangle = \langle \eta|_{\rho} \rangle$  and  $\{\langle \eta_1|_{\rho} \rangle, \dots, \langle \eta_k|_{\rho} \rangle\}$ , being a collection of primes for  $\eta|_{\rho}$ , is disjoint. By assumption  $\langle \text{Sdds}_{\alpha|_{\rho}}(v_i|_{\mathbf{A}}) \rangle$  is disjoint, so for any other  $\eta'|_{\rho} = \{(\eta'_1|_{\rho}, \top), \dots, (\eta'_{k'}|_{\rho}, \top)\} \in \text{Sdds}_{\alpha|_{\rho}}(v_i|_{\mathbf{A}})$  distinct from  $\eta|_{\rho}$ , we have  $\langle \eta|_{\rho} \rangle \wedge \langle \eta'|_{\rho} \rangle = \perp$ . Then for any  $i \in [k]$  and  $j \in [k']$ , we have  $\langle \eta_i|_{\rho} \rangle \wedge \langle \eta'_j|_{\rho} \rangle = \perp$ . Thus  $\langle \text{Sdds}_{\alpha|_{\rho}}(v_{i+1}|_{\mathbf{A}}) \rangle$  is disjoint.  $\square$

**Theorem 3.5.** *Let  $\alpha$  be an SDD of size  $s$  that respects a vtree  $\mathbf{v}$  and suppose that it computes the function  $f : \{0, 1\}^{\mathbf{X}} \rightarrow \{0, 1\}$ . Suppose that  $\{\mathbf{A}, \mathbf{B}\}$  is a shell partition for  $\mathbf{X}$  and that  $\mathbf{A}$  is the shell. Let  $b$  be the vertex in  $\mathbf{v}$  for which  $\text{Vars}(b) = \mathbf{B}$  and  $\text{Vars}(\text{Parent}(b)) \not\subseteq \mathbf{B}$ .*

*Consider the communication game where Alice has the variables  $\mathbf{A}$ , Bob has the variables  $\mathbf{B}$ , and they are trying to compute  $f(\mathbf{A}, \mathbf{B})$ . There is a log  $s$ -bit unambiguous communication protocol computing  $f$ .*

*Proof.* Suppose that Alice and Bob both know the SDD  $\alpha$ . Let  $\rho : \mathbf{A} \rightarrow \{0, 1\}$  be the partial assignment corresponding to Alice's input. This is a shell restriction. Alice may then privately construct the pruned SDD  $\alpha|_{\rho}$ , which computes  $f|_{\rho}$  by Proposition 3.3. Further,  $\alpha|_{\rho}$  evaluates to 1 under Bob's input  $\phi : \mathbf{B} \rightarrow \{0, 1\}$  if and only if there exists a pruned-SDD  $\eta|_{\rho} \in \text{Sdds}_{\alpha|_{\rho}}(b|_{\mathbf{A}})$  such that  $\langle \eta|_{\rho} \rangle(\phi) = 1$ .

By Proposition 3.4,  $\langle \text{Sdds}_{\alpha|_{\rho}}(b|_{\mathbf{A}}) \rangle$  is disjoint. Also, since  $\rho$  is a shell restriction with shell  $\mathbf{A}$ , and  $\text{Vars}(b) = \mathbf{B} = \mathbf{X} \setminus \mathbf{A}$ , every SDD in  $\text{Sdds}_{\alpha|_{\rho}}(b|_{\mathbf{A}})$  was unchanged by  $\rho$ . In particular, this means that  $\text{Sdds}_{\alpha|_{\rho}}(b|_{\mathbf{A}}) \subseteq \text{Sdds}_{\alpha}(b)$  and any pruned-SDD  $\eta|_{\rho}$  can be viewed as some  $\eta \in \text{Sdds}_{\alpha}(b)$  that is also in  $\text{Sdds}_{\alpha|_{\rho}}(b|_{\mathbf{A}})$ .

For the protocol Alice nondeterministically selects an  $\eta$  from  $\text{Sdds}_{\alpha|_{\rho}}(b|_{\mathbf{A}})$  and then sends its identity as a member of  $\text{Sdds}_{\alpha}(b)$  to Bob. This requires at most  $\log s$  bits. Bob will output 1 on his input  $\phi$  if and only if  $\langle \eta \rangle(\phi) = 1$ , which he can test since he knows  $\alpha$  and  $b$ . This protocol is unambiguous since the fact that  $\langle \text{Sdds}_{\alpha|_{\rho}}(b|_{\mathbf{A}}) \rangle$  is disjoint means that for any input  $\phi$  to Bob there is at most one  $\eta \in \text{Sdds}_{\alpha|_{\rho}}(b|_{\mathbf{A}})$  such that  $\langle \eta \rangle(\phi) = 1$ . Since Bob knows  $\alpha$ , he also knows  $\eta$  and can therefore compute  $\langle \eta \rangle(\phi)$ . Since  $\alpha$  computes  $f$ , if  $\langle \eta \rangle(\phi) = 1$  then  $f(\phi, \rho) = 1$ . Otherwise all of the functions in  $\langle \text{Sdds}_{\alpha|_{\rho}}(b|_{\mathbf{A}}) \rangle$  evaluate to 0 on input  $\phi$  so  $f(\phi, \rho) = 0$ .  $\square$

We can relate the deterministic and unambiguous communication complexities of a function using the following result from [Yannakakis, 1991]. We include a proof of this result in the appendix for completeness.

**Theorem 3.6** (Yannakakis). *If there is an  $g$ -bit unambiguous communication protocol for a function  $f : \{0, 1\}^{\mathbf{A}} \times \{0, 1\}^{\mathbf{B}} \rightarrow \{0, 1\}$ , then there is a  $(g + 1)^2$ -bit deterministic protocol for  $f$ .*

The following 1/3-2/3 lemma is standard.

**Lemma 3.7.** *For a vtree  $\mathbf{v}$  for  $L$  variables, if a vertex  $b$  satisfies  $\frac{1}{3}L \leq |\text{Vars}(b)| \leq \frac{2}{3}L$ , we call it a (1/3, 2/3) vertex. Every vtree contains a (1/3, 2/3) vertex.*

**Definition 3.8.** Let  $\mathbf{X}$  be a set of variables and  $(\mathbf{A}, \mathbf{B})$  a partition of  $\mathbf{X}$ . We call the partition  $(\mathbf{A}, \mathbf{B})$  a  $(\delta, 1 - \delta)$ -partition for  $\delta \in [0, 1/2]$  if  $\min(|\mathbf{A}|, |\mathbf{B}|) \geq \delta|\mathbf{X}|$ . That is, the minimum size of one side of the partition is at least a  $\delta$ -fraction of the total number of variables.

The best  $(\delta, 1 - \delta)$ -partition communication complexity of a Boolean function  $f : \{0, 1\}^{\mathbf{X}} \rightarrow \{0, 1\}$  is  $\min(CC(f(\mathbf{A}, \mathbf{B})))$  where the minimum is taken over all  $(\delta, 1 - \delta)$ -partitions  $(\mathbf{A}, \mathbf{B})$ .

**Theorem 3.9.** If the best  $(1/3, 2/3)$ -partition communication complexity of a Boolean function  $f : \{0, 1\}^{\mathbf{X}} \rightarrow \{0, 1\}$  is  $C$ , then an SDD computing  $f$  has size at least  $2^{\sqrt{C}-1}$ .

*Proof.* Suppose that  $\alpha$  is an SDD of size  $s$  respecting the vtree  $\mathbf{v}$  for variables  $\mathbf{X}$ , and that  $\alpha$  computes  $f$ . From Lemma 3.7 the vtree  $\mathbf{v}$  contains a  $(1/3, 2/3)$  vertex  $b$ . This  $(1/3, 2/3)$  vertex  $b$  induces a  $(1/3, 2/3)$ -partition of the variables  $\{\mathbf{A}, \mathbf{B}\}$  where  $\mathbf{B} = \text{Vars}(b)$  and  $\mathbf{A} = \text{Shell}(b)$ . Further, this partition  $\{\mathbf{A}, \mathbf{B}\}$  is a shell partition. By Theorem 3.5, there exists a  $\log s$ -bit unambiguous communication protocol for  $f(\mathbf{A}, \mathbf{B})$ . Then by Theorem 3.6, there exists a  $(\log(s) + 1)^2$ -bit deterministic communication protocol for  $f(\mathbf{A}, \mathbf{B})$ . Since the best  $(1/3, 2/3)$ -partition communication complexity of  $f$  is  $C$ , we have that  $C \leq (\log(s) + 1)^2$  which implies that  $s \geq 2^{\sqrt{C}-1}$  as stated.  $\square$

## 4 Lower Bounds for SDDs

There are a large number of predicates  $f : \{0, 1\}^n \rightarrow \{0, 1\}$  for which the  $(1/3, 2/3)$ -partition communication complexity is  $\Omega(n)$  and by Theorem 3.9 each of these requires SDD size  $2^{\Omega(\sqrt{n})}$ . The usual best-partition communication complexity is  $(1/2, 1/2)$ -partition communication complexity. For example, the function `SHIFT-EDEQ` which takes as inputs  $x, y \in \{0, 1\}^n$  and  $z \in \{0, 1\}^{\lceil \log_2 n \rceil}$  and tests whether or not  $y = \text{SHIFT}(x, z)$  where  $\text{SHIFT}(x, z)$  is the cyclic shift of  $x$  by  $(z)_2$  positions. However, as is typical of these functions, the same proof which shows that the  $(1/2, 1/2)$ -partition communication complexity of `SHIFTEDEQ` is  $\Omega(n)$  also shows that its  $(1/3, 2/3)$ -partition communication complexity is  $\Omega(n)$ . However, most of these functions are not typical of predicates to which one might want to apply weighted model counting. Instead we analyze SDDs for formulas derived from a natural class of database queries. We are able to characterize SDD size for these queries, proving exponential lower bounds for every such query that cannot already be represented in linear size by an OBDD. This includes an example of a query called  $Q_V$  for which FBDDs are polynomial size but the best SDD requires exponential size.

### 4.1 SDD Knowledge Compilation for Database Query Lineages

We analyze SDDs for a natural class of database queries called the *union of conjunctive queries (UCQ)*. This includes all queries given by the grammar

$$q ::= R(\mathbf{x}) \mid \exists x q \mid q \wedge q \mid q \vee q$$

where  $R(\mathbf{x})$  is an elementary relation and  $x$  is a variable. For each such query  $q$ , given an input database  $D$ , the query's *lineage*,  $\Phi_q^D$ , is a Boolean expression for  $q$  over Boolean variables that correspond to tuples in  $D$ . In general, one thinks of the query size as fixed and considers the complexity of query evaluation as a function of the size of the database. The following formulas are lineages (or parts thereof) of well-known queries that are fundamental for probabilistic databases [Dalvi and Suciu, 2012, Jha and Suciu, 2013]

over a particular database  $D_0$  (called the *complete bipartite graph of size  $m$*  in [Jha and Suciu, 2013]):

$$\begin{aligned}
H_0 &= \bigvee_{i,j \in [m]} R_i S_{ij} T_j \\
Q_V &= \bigvee_{i,j \in [m]} R_i S_{ij} \vee S_{ij} T_j \vee R_i T_j \\
H_1 &= \bigvee_{i,j \in [m]} R_i S_{ij} \vee S_{ij} T_j \\
H_{k0} &= \bigvee_{i \in [m]} R_i S_{ij}^1 \quad \text{for } k \geq 1 \\
H_{k\ell} &= \bigvee_{i,j \in [m]} S_{ij}^\ell S_{ij}^{\ell+1} \quad \text{for } 0 < \ell < k \\
H_{kk} &= \bigvee_{i \in [m]} S_{ij}^k T_j \quad \text{for } k \geq 1.
\end{aligned}$$

(The corresponding queries are represented using lower case letters  $h_0, q_V, h_1, h_{k0}, \dots, h_{kk}$  and involve unary relations  $R$  and  $T$ , as well as binary relations  $S$  and  $S^k$ . For example,  $h_0 = \exists x_0 \exists y_0 R(x_0)S(x_0, y_0)T(y_0)$ .) The following lemma will be useful in identifying subformulas of the above query lineages that can be used to compute the set disjointness function.

**Proposition 4.1.** *Let the elements of  $[m] \times [m]$  be partitioned into two sets  $A$  and  $B$ , each of size at least  $\delta m^2$ . Let  $\text{Row}(i)$  denote  $\{i\} \times [m]$  and  $\text{Col}(j)$  denote  $[m] \times \{j\}$ . Define  $W_{\text{Row}} = \{i \in [m] \mid \emptyset \neq \text{Row}(i) \cap A \text{ and } \emptyset \neq \text{Row}(i) \cap B\}$ . That is,  $\text{Row}(i)$  for  $i \in W_{\text{Row}}$  is split into two nonempty pieces by the partition. Similarly, define  $W_{\text{Col}} = \{j \in [m] \mid \emptyset \neq \text{Col}(j) \cap A \text{ and } \emptyset \neq \text{Col}(j) \cap B\}$ . Then*

$$\max(|W_{\text{Row}}|, |W_{\text{Col}}|) \geq \sqrt{\delta} \cdot m.$$

*Proof.* Suppose that both  $|W_{\text{Row}}| < m$  and  $|W_{\text{Col}}| < m$ . By definition, if  $i \notin W_{\text{Row}}$  then one of  $A$  or  $B$  contains an entire row,  $\text{Row}(i)$ , say  $A$  without loss of generality. This implies that no column  $\text{Col}(j)$  is entirely contained in  $B$ . Since  $|W_{\text{Col}}| < m$ , there is some column  $\text{Col}(j)$  that is entirely contained in  $A$ . This in turn implies that  $B$  does not contain any full row. In particular, we have that  $A$  contains all rows in  $[m] \setminus W_{\text{Row}}$  and all columns in  $[m] \setminus W_{\text{Col}}$  and thus  $B \subseteq W_{\text{Row}} \times W_{\text{Col}}$  and so  $|B| \leq |W_{\text{Row}}| \cdot |W_{\text{Col}}|$ . By assumption,  $|B| \geq \delta m^2$ . Hence  $|W_{\text{Row}}| \cdot |W_{\text{Col}}| \geq \delta m^2$  and so  $\max\{|W_{\text{Row}}|, |W_{\text{Col}}|\} \geq \sqrt{\delta} \cdot m$ .  $\square$

**Theorem 4.2.** *For  $m \geq 6$ , the best  $(1/3, 2/3)$ -partition communication complexity of  $Q_V$ ,  $H_0$ , and of  $H_1$  is at least  $m/3$ .*

*Proof.* Let  $\mathbf{X}$  be the set of variables appearing in  $Q_V$  (or  $H_1$ ) and let  $(\mathbf{A}, \mathbf{B})$  be a  $(1/3, 2/3)$ -partition of  $\mathbf{X}$ . Let  $(A, B)$  be the partition of  $[m] \times [m]$  induced by  $(\mathbf{A}, \mathbf{B})$  and define  $W_{\text{Row}}$  and  $W_{\text{Col}}$  as in Proposition 4.1. Since  $|\mathbf{X}| = m^2 + 2m$  and only elements of  $[m] \times [m]$  are relevant,  $|A|, |B| \geq (m^2 + 2m)/3 - 2m = (1 - 4/m)m^2/3 \geq m^2/9$  for  $m \geq 6$  and hence  $\max(|W_{\text{Row}}|, |W_{\text{Col}}|) \geq m/3$ . We complete the proof by showing that computing  $Q_V(\mathbf{A}, \mathbf{B})$  and  $H_1(\mathbf{A}, \mathbf{B})$  each require at least  $\max(|W_{\text{Row}}|, |W_{\text{Col}}|)$  bits of communication between Alice and Bob. We will do this by showing that for a particular subset of inputs,  $Q_V$  is equivalent to the disjointness function for a  $\max(|W_{\text{Row}}|, |W_{\text{Col}}|)$  size set.

Suppose without loss of generality that  $|W_{\text{Row}}| \geq |W_{\text{Col}}|$ . Set all  $T_j = 0$  and for each  $i \notin W_{\text{Row}}$  set  $R_i = 0$ . For each  $i \in W_{\text{Row}}$  for which  $R_i \in \mathbf{A}$ , set all  $S_{ij} \in \mathbf{A}$  to 0, let  $j_i$  be minimal such that  $S_{ij_i} \in \mathbf{B}$ , and set  $S_{ij} \in \mathbf{B}$  to 0 for all  $j > j_i$ . (Such an index  $j_i$  must exist since  $i \in W_{\text{Row}}$ .) Similarly, For each  $i \in W_{\text{Row}}$  for which  $R_i \in \mathbf{B}$ , set all  $S_{ij} \in \mathbf{B}$  to 0, let  $j_i$  be minimal such that  $S_{ij_i} \in \mathbf{A}$ , and set  $S_{ij} \in \mathbf{A}$  to 0 for all  $j > j_i$ . In particular, under this partial assignment, we have

$$Q_V = H_1 = \bigvee_{i \in W_{\text{Row}}} R_i S_{ij_i}$$

and for each  $i \in W_{\text{Row}}$ , Alice holds one of  $R_i$  or  $S_{ij_i}$  and Bob holds the other. We can reduce  $H_0$  to the same quantity by setting all  $T_j = 1$ . This is precisely the set disjointness problem on two sets of size  $|W_{\text{Row}}|$  where membership of  $i$  in each player's set is determined by the value of the unset bit indexed by  $i$  that player holds. Therefore, computing  $Q_V$  or  $H_1$  requires at least  $|W_{\text{Row}}|$  bits of communication, as desired.  $\square$

Combining this with Theorem 3.9, we immediately obtain the following:

**Theorem 4.3.** *For  $m \geq 6$ , any SDD representing  $Q_V$  or  $H_1$  requires size at least  $2\sqrt{m/3}-1$ .*

As [Jha and Suciu, 2013] has shown that  $Q_V$  has FBDD size  $O(m^2)$ , we obtain the following separation.

**Corollary 4.4.** *FBDDs can be exponentially more succinct than SDDs. In particular,  $Q_V$  has FBDD size  $O(m^2)$  but every SDD for  $Q_V$  requires size  $2\sqrt{m/3}-1$  for  $m \geq 6$ .*

We now consider the formulas  $H_{ki}$  above. Though they seem somewhat specialized, these formulas are fundamental to UCQ queries: [Jha and Suciu, 2013] define the notion of an *inversion* in a UCQ query and use it to characterize the OBDD size of UCQ queries. In particular they show that if a query  $q$  is *inversion-free* then the OBDD size of its lineage  $Q$  is linear and if  $q$  has an minimum inversion length  $k \geq 1$  then it requires OBDD size  $2^{\Omega(n/k)}$  where  $n$  is the domain size of all attributes. Jha and Suciu obtain this lower bound by analyzing the  $H_{ki}$  we defined above. (We will not define the notion of inversions, or their lengths, and instead use the definition as a black box. However, as an example, the query associated with  $H_1$  has an inversion of length 1 so its OBDD size is  $2^{\Omega(m)}$ .)

**Proposition 4.5.** *[Jha and Suciu, 2013] Let  $q$  be a query with a length  $k \geq 1$  inversion. Let  $D_0$  be the complete bipartite graph of size  $m$ . There exists a database  $D$  for  $q$ , along with variable restrictions  $\rho_i$  for all  $i \in [0, k]$ , such that  $|D| = O(|D_0|)$  and  $\Phi_q^D|_{\rho_i} = \Phi_{h_{ki}}^{D_0} = H_{ki}$*

**Theorem 4.6.** *Let  $k \geq 2$  and assume that  $m \geq 6$ . Let  $q$  be a query with a length  $k \geq 2$  inversion. Then there exists a database  $D$  for which any SDD for  $Q = \Phi_q^D$  has size at least  $2\sqrt{m/k/3}-1$ .*

*Proof.* Given a query  $q$ , let  $D$  be the database for  $q$  constructed in Proposition 4.5. Fix the vtree  $\mathbf{v}$  over  $\mathbf{X}_k$  respected by an SDD  $\alpha$  for  $\Phi_q^D$ . By Lemma 3.7, there exists a  $(1/3, 2/3)$  node  $b$  in the vtree  $\mathbf{v}$  that gives a  $(1/3, 2/3)$  partition  $\{\mathbf{A}, \mathbf{B}\}$  of  $\mathbf{X}_k$ . By Proposition 4.5, there are restrictions  $\rho_0, \dots, \rho_k$  such that  $\Phi_q^D|_{\rho_i} = H_{ki}$  for all  $i$ . Thus  $\alpha|_{\rho_i}$  is a (pruned) SDD, of size  $\leq$  that of  $\alpha$ , respecting  $\mathbf{v}|_{\rho_i}$  and computing  $H_{ki}$ . Observe that the restriction of  $\{\mathbf{A}, \mathbf{B}\}$  to the variables of  $\mathbf{X}_{ki}$  is also shell partition of  $\mathbf{v}|_{\rho_i}$  at node  $b$ .

We will show that there must exist an  $H_{ki}$  for which  $\text{CC}(H_{ki}(\mathbf{A}, \mathbf{B})) \geq m/(9k)$  and therefore by Theorem 3.6, this implies that the unambiguous communication complexity of  $H_{ki}$  is at least  $\frac{1}{3}\sqrt{m/k} - 1$ . Then by Theorem 3.5, any SDD respecting  $\mathbf{v}$  that computes  $H_{ki}$  has size at least  $2^{\frac{1}{3}\sqrt{m/k}-1}$ .

Let  $W_{\text{Chain}}$  contain all pairs  $(i, j)$  for which both  $\mathbf{A} \cap \bigcup_{\ell=1}^k \{S_{ij}^\ell\} \neq \emptyset$  and  $\mathbf{B} \cap \bigcup_{\ell=1}^k \{S_{ij}^\ell\} \neq \emptyset$  and Let  $\gamma = 1/9$ . We will consider two cases: either  $|W_{\text{Chain}}| \geq \gamma \cdot m$  or  $|W_{\text{Chain}}| < \gamma \cdot m$ .

In the first case, since  $|W_{\text{Chain}}| \geq \gamma \cdot m$ , there must exist at least  $\gamma \cdot m$  tuples  $(i, j, \ell)$  for which either  $S_{ij}^\ell \in \mathbf{A}$  and  $S_{ij}^{\ell+1} \in \mathbf{B}$  or vice-versa. Call the set of these tuples  $\mathbf{T}$ . Then, since there are  $k-1$  choices of  $\ell < k$ , there exists some  $\ell^*$  such that the set  $\mathbf{T}_{\ell^*} := \mathbf{T} \cap [m] \times [m] \times \{\ell^*\}$  contains at least  $\gamma \cdot m/(k-1) > m/(9k)$  elements. If we set all variables of  $\mathbf{X}_{k\ell^*}$  outside of  $\mathbf{T}_{\ell^*}$  to 0, the function  $H_{k\ell^*}$  corresponds to solving a disjointness problem between Alice and Bob on the elements of  $\mathbf{T}_{\ell^*}$ . Thus the communication complexity of  $H_{k\ell^*}$  under the partition  $\{\mathbf{A}, \mathbf{B}\}$  is at least  $m/(9k)$ .

In the second case, consider the largest square submatrix  $M$  of  $[m] \times [m]$  that does not contain any member of  $W_{\text{Chain}}$ . We mimic the argument of Theorem 4.2 on this submatrix  $M$ . By definition,  $M$  has side  $m' \geq (1-\gamma)m$ . For every  $(i, j)$  in  $M$ , either  $\mathbf{A}$  or  $\mathbf{B}$  contains all  $S_{ij}^\ell$ ; let  $A$  be those  $(i, j)$  such that these are in  $\mathbf{A}$  and  $B$  be those  $(i, j)$  for which they are in  $\mathbf{B}$ . Since  $|\mathbf{A}|, |\mathbf{B}| \geq |\mathbf{X}_k|/3 = (km^2 + 2m)/3$  and there are at most  $2m + (\gamma^2 + 2\gamma)km^2$  variables not in  $M$ ,

$$\begin{aligned} |A|, |B| &\geq [(km^2 + 2m)/3 - 2m + (\gamma^2 + 2\gamma)km^2]/k \\ &= [(1-\gamma)^2 - 2/3 - 4/(3km)]m^2 > (m/18)^2 \end{aligned}$$

since  $k \geq 2$ . Applying Proposition 4.1, we see that  $\max(|W_{\text{Row}}|, |W_{\text{Col}}|) \geq m/18 \geq m/(9k)$ . By the same argument presented in the proof of Theorem 4.2, we have both  $\text{CC}(H_{k0}(\mathbf{A}, \mathbf{B})) \geq |W_{\text{Row}}|$  and  $\text{CC}(H_{kk}(\mathbf{A}, \mathbf{B})) \geq |W_{\text{Col}}|$  so at least one of these is at least  $m/(9k)$  and the theorem follows.  $\square$

It follows that for inversion-free UCQ queries, both SDD and OBDD sizes of any lineage are linear, while UCQ queries with inversions (of length  $k$ ) have worse-case lineage size that is exponential ( $2^{\Omega(m/k)}$  for OBDDs and  $2^{\Omega(\sqrt{m/k})}$  for SDDs). Note that the same SDD size lower bound for UCQ query lineage  $Q = \Phi_q^D$  applies to its dual  $Q^* = \Phi_q^D$  as follows: Flipping the signs on the variables in  $Q^*$  yields a function equivalent to  $\neg Q$ . So flipping the variable signs at the leaves of an SDD for  $Q^*$  we obtain an SDD of the same size for  $\neg Q$  and hence a deterministic protocol that also can compute  $Q$ .

## 5 Simulating DNNFs by OR-FBDDs

In this section, we extend the simulation of decision-DNNFs by FBDDs from [Beame et al., 2013] to obtain a simulation of general DNNFs by OR-FBDDs with at most a quasipolynomial increase in size. This simulation yields lower bounds on DNNF size from OR-FBDD lower bounds. This simulation is also tight, since [Razgon, 2015a, Razgon, 2014] has shown a quasipolynomial separation between the sizes of DNNFs and OR-FBDDs.

**Definition 5.1.** For each AND node  $u$  in a DNNF  $\mathcal{D}$ , let  $M_u$  be the number of AND nodes in the subgraph  $D_u$ . We call  $u$ 's left child  $u_l$  and its right child  $u_r$ . We will assume  $M_{u_l} \leq M_{u_r}$  (otherwise we swap  $u_l$  and  $u_r$ ).

For each AND node  $u$ , we classify the edge  $(u, u_l)$  as a light edge and the edge  $(u, u_r)$  a heavy edge. We classify every other edge in  $\mathcal{D}$  as a neutral edge.

For a DNNF  $\mathcal{D}$  or an OR-FBDD  $\mathcal{F}$ , we denote the functions that  $\mathcal{D}$  and  $\mathcal{F}$  compute as  $\Phi_{\mathcal{D}}$  and  $\Phi_{\mathcal{F}}$ .

### Constructing the OR-FBDD

For a DNNF  $\mathcal{D}$ , we will treat a leaf labeled by the variable  $X$  as a decision node that points to a 0-sink node if  $X = 0$  and a 1-sink node if  $X = 1$ , and vice-versa for a leaf labeled by  $\neg X$ . We also assume that each AND node has just two children, which only affects the DNNF size by at most polynomially.

**Definition 5.2.** Fix a DNNF  $\mathcal{D}$ . For a node  $u$  in  $\mathcal{D}$  and a path  $P$  from the root to  $u$ , let  $S(P)$  be the set of light edges along  $P$  and  $S(u) = \{S(P) \mid P \text{ is a path from the root to } u\}$ .

We will construct an OR-FBDD  $\mathcal{F}$  that computes the same boolean function as  $\mathcal{D}$ . Its nodes are pairs  $(u, s)$  where  $u$  is a node in  $\mathcal{D}$  and the set of light edges  $s$  belongs to  $S(u)$ . Its root is  $(\text{root}(\mathcal{D}), \emptyset)$ . The edges in  $\mathcal{F}$  are of three types:

Type 1: For each light edge  $e = (u, v)$  in  $\mathcal{D}$  and  $s \in S(u)$ , add the edge  $((u, s), (v, s \cup \{e\}))$  to  $\mathcal{F}$ .

Type 2: For each neutral edge  $e = (u, v)$  in  $\mathcal{D}$  and  $s \in S(u)$ , add the edge  $((u, s), (v, s))$  to  $\mathcal{F}$ .

Type 3: For each heavy edge  $(u, v_r)$ , let  $e = (u, v_l)$  be its sibling light edge. For each  $s \in S(u)$  and 1-sink node  $w$  in  $D_{v_l}$ , add the edge  $((w, s \cup \{e\}), (v_r, s))$  to  $\mathcal{F}$ .

We label the nodes  $u' = (u, s)$  as follows: (1) if  $u$  is a decision node in  $\mathcal{D}$  for the variable  $X$  then  $u'$  is a decision node in  $\mathcal{F}$  testing the same variable  $X$ , (2) if  $u$  is an AND-node, then  $u'$  is a no-op node, (3) if  $u$  is an OR node it remains an OR node. (4) if  $u$  is a 0-sink node, then  $u'$  is a 0-sink node, (5) if  $u$  is a 1-sink node, then: if  $s = \emptyset$  then  $u'$  is a 1-sink node, otherwise it is a no-op node.

We show an example of this construction in Figure 3.

### Size and Correctness

**Lemma 5.3.** For the DNNF  $\mathcal{D}$  let  $L$  denote the maximum number of light edges from the root to a leaf,  $M$  the number of AND nodes and  $N$  the total number of nodes. Then  $\mathcal{F}$  has at most  $NM^L$  nodes. Further, this is  $N \cdot 2^{\log^2 N}$ .

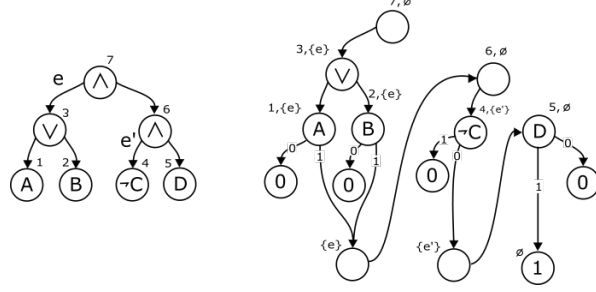


Figure 3: A DNNF and our construction of an equivalent OR-FBDD.

*Proof.* The nodes in  $\mathcal{F}$  are labeled  $(u, s)$ . There are  $N$  possible nodes  $u$  and at most  $M^L$  choices for the set  $s$ , as each path to  $u$  has at most  $L$  light edges.

Consider a root to leaf path with  $L$  light edges. As we traverse this path, every time we cross a light edge, we decrease the number of descendant AND nodes by more than half. Thus we must have begun with more than  $2^L$  descendant AND nodes at the root so that  $N \geq M > 2^L$ . This implies that  $NM^L$  is quasipolynomial in  $N$ ,

This upper bound is quasipolynomial in  $N$ , we will show that  $M > 2^L$ . Then, since  $N \geq M$ ,  $NM^L \leq N2^{\log^2 M} \leq N2^{\log^2 N}$ .  $\square$

The proof of the following lemma is in the full paper.

**Lemma 5.4.**  $\mathcal{F}$  is a correct OR-FBDD with no-op nodes that computes the same function as  $\mathcal{D}$ .

Using the quasipolynomial simulation of DNNFs by OR-FBDDs, we obtain DNNF lower bounds from OR-FBDD lower bounds.

**Definition 5.5.** Function  $\text{PERM}_n$  takes an  $n \times n$  boolean matrix  $M$  as input and outputs 1 if and only if  $M$  is a permutation matrix. The function  $\text{ROW-COL}_n$  takes an  $n \times n$  boolean matrix  $M$  as input and outputs 1 if and only if  $M$  has an all-0 row or an all-0 column.

**Theorem 5.6.** Any OR-FBDD computing  $\text{PERM}_n$  or  $\text{ROW-COL}$ , must have size  $2^{\Omega(n)}$  [Wegener, 2000].

**Corollary 5.7.** Any DNNF computing  $\text{PERM}_n$  or  $\text{ROW-COL}$  has size at least  $2^{\Omega(\sqrt{n})}$

## 6 Discussion

We have made the first significant progress in understanding the complexity of general DNNF representations. We have also provided a new connection between SDD representations and best-partition communication complexity. Best-partition communication complexity is a standard technique used to derive lower bounds on OBDD size, where it often yields asymptotically tight results. For communication lower bound  $C$ , the lower bound for OBDD size is  $2^C$  and the lower bound we have shown for SDD size is  $2^{\sqrt{C}} - 1$ . This is a quasipolynomial difference and matches the quasipolynomial separation between OBDD and SDD size shown in [Razgon, 2014]. Is there always a quasipolynomial simulation of SDDs by OBDDs in general, matching the quasipolynomial simulation of decision-DNNFs by FBDDs? Our separation result shows an example for which SDDs are sometimes exponentially less concise than FBDDs, and hence decision-DNNFs also. Are SDDs ever more concise than decision-DNNFs?

By plugging in the arguments of [Pipatsrisawat and Darwiche, 2010, Pipatsrisawat, 2010] in place of Theorem 3.5, all of our lower bounds immediately extend to size lower bounds for structured deterministic DNNFs (d-DNNFs), of which SDDs are a special case. It remains open whether structured d-DNNFs are strictly more concise than SDDs. [Pipatsrisawat and Darwiche, 2008, Pipatsrisawat, 2010] have proved an exponential separation between structured d-DNNFs and OBDDs using the *Indirect Storage Access (ISA)* function [Breitbart et al., 1995], but the small structured d-DNNF for this function is very far from an SDD.

It is immediate that, under any variable partition, the  $ISA_n$  function has an  $O(\log n)$ -bit two-round deterministic communication protocol. On the other hand, efficient one-round (i.e., one-way) communication protocols yield small OBDDs so there are two possibilities if SDDs and structured d-DNNFs have different power. Either (1) communication complexity considerations on their own are not enough to derive a separation between SDDs and structured d-DNNFs, or (2) every SDD can be simulated by an efficient one-way communication protocol, in which case SDDs can be simulated efficiently by OBDDs (though the ordering cannot be the same as the natural traversal of the associated vtree, as shown by [Xue et al., 2012]).

## Acknowledgements

We thank Dan Suciu and Guy Van den Broeck for helpful comments and suggestions.

## References

- [Beame et al., 2013] Beame, P., Li, J., Roy, S., and Suciu, D. (2013). Lower bounds for exact model counting and applications in probabilistic databases. In *UAI*, pages 157–162.
- [Beame et al., 2014] Beame, P., Li, J., Roy, S., and Suciu, D. (2014). Counting of query expressions: Limitations of propositional methods. In *ICDT*, pages 177–188.
- [Breitbart et al., 1995] Breitbart, Y., Hunt III, H. B., and Rosenkrantz, D. J. (1995). On the size of binary decision diagrams representing boolean functions. *Theor. Comput. Sci.*, 145(1&2):45–69.
- [Dalvi and Suciu, 2012] Dalvi, N. N. and Suciu, D. (2012). The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6):30.
- [Darwiche, 2001] Darwiche, A. (2001). Decomposable negation normal form. *J. ACM*, 48(4):608–647.
- [Darwiche, 2011] Darwiche, A. (2011). SDD: A new canonical representation of propositional knowledge bases. In *IJCAI 2011*, pages 819–826.
- [Gomes et al., 2009] Gomes, C. P., Sabharwal, A., and Selman, B. (2009). Model counting. In *Handbook of Satisfiability*, pages 633–654. IOS Press.
- [Huang and Darwiche, 2007] Huang, J. and Darwiche, A. (2007). The language of search. *JAIR*, 29:191–219.
- [Jha and Suciu, 2013] Jha, A. K. and Suciu, D. (2013). Knowledge compilation meets database theory: Compiling queries to decision diagrams. *Theory Comput. Syst.*, 52(3):403–440.
- [Kushilevitz and Nisan, 1997] Kushilevitz, E. and Nisan, N. (1997). *Communication Complexity*. Cambridge University Press, Cambridge, England ; New York.
- [Pipatsrisawat and Darwiche, 2008] Pipatsrisawat, K. and Darwiche, A. (2008). New compilation languages based on structured decomposability. In *AAAI*, pages 517–522.
- [Pipatsrisawat and Darwiche, 2010] Pipatsrisawat, K. and Darwiche, A. (2010). A lower bound on the size of Decomposable Negation Normal Form. In *AAAI*, pages 345–350.
- [Pipatsrisawat, 2010] Pipatsrisawat, T. (2010). *Reasoning with Propositional Knowledge: Frameworks for Boolean Satisfiability and Knowledge Compilation*. PhD thesis, UCLA.
- [Razgon, 2014] Razgon, I. (2014). On obdds for cnfs of bounded treewidth. In Baral, C., Giacomo, G. D., and Eiter, T., editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. AAAI Press.

- [Razgon, 2015a] Razgon, I. (2015a). On the read-once property of branching programs and CNFs of bounded treewidth. *CoRR*, abs/1411.0264v3.
- [Razgon, 2015b] Razgon, I. (2015b). Quasipolynomial simulation of DNNF by a non-deterministic read-once branching program. In Pesant, G., editor, *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, volume 9255 of *Lecture Notes in Computer Science*, pages 367–375. Springer.
- [SDD, 2014] SDD (2014). The SDD Package: Version 1.1.1. <http://reasoning.cs.ucla.edu/sdd/>.
- [Van den Broeck and Darwiche, 2015] Van den Broeck, G. and Darwiche, A. (2015). On the role of canonicity in knowledge compilation. In *AAAI*, pages 1641–1648.
- [Wegener, 2000] Wegener, I. (2000). *Branching programs and binary decision diagrams: theory and applications*. SIAM, Philadelphia, PA, USA.
- [Xue et al., 2012] Xue, Y., Choi, A., and Darwiche, A. (2012). Basing decisions on sentences in decision diagrams. In *AAAI*, pages 842–849.
- [Yannakakis, 1991] Yannakakis, M. (1991). Expressing combinatorial optimization problems by linear programs. *Journal of Computer and System Sciences*, 43(3):441–466.

## A Proof of Theorem 3.6

Let  $f(\mathbf{A}, \mathbf{B})$  be a function with unambiguous communication complexity  $g$  and let  $M_f$  be its communication matrix. Then there exists a set  $D$  of  $2^g$  disjoint monochromatic rectangles that cover the 1’s of  $M_f$ .

Let  $G$  be a graph whose nodes are the rectangles in  $D$  and which has an edge connecting two rectangles if they share some row of  $M_f$ . Then each row  $r$  of  $M_f$  corresponds to a clique  $K_r$  containing the rectangles intersecting  $r$ . Similarly, every column  $c$  corresponds to an independent set  $I_c$  containing the rectangles intersecting  $c$ . For each row  $r$  and column  $c$ , the corresponding entry  $M_f(r, c)$  is 1 if and only if  $K_r \cap I_c \neq \emptyset$ . Thus for proving the theorem, it suffices to give a  $g^2$  deterministic protocol for solving the Clique vs Independent set problem on a graph  $G$  with  $2^g$  vertices.

The protocol reduces the graph in each step. Suppose that Alice holds a clique  $K$  of an  $n$  vertex graph  $G$  and Bob holds an independent set  $I$ . In each round Alice sends a node  $u \in K$  that is adjacent to fewer than half the nodes of  $G$ , or if no such node exists, she notifies Bob.

If Alice sent the node  $u$ , then Bob responds with whether (i)  $u \in I$ , in which case  $K \cap I \neq \emptyset$ , or (ii) that  $u$  is not adjacent to any node of  $I$ , in which case  $K \cap I = \emptyset$ . If neither (i) nor (ii) occur then the nodes not adjacent to  $u$  are removed from  $G$  as they cannot be in  $K$  and the protocol repeats.

Otherwise, if every  $u \in K$  is adjacent to over half the nodes of  $G$ , Bob sends a node  $v \in I$  that is adjacent to at least half the nodes in  $G$  if such a  $v$  exists. In this case Alice tells Bob that (i)  $v \in K$  so that  $K \cap I \neq \emptyset$ , or (ii)  $v$  is adjacent to all nodes in  $K$  so that  $K \cap I = \emptyset$ . Otherwise, Bob says he has no such  $v \in I$  and the nodes adjacent to  $v$  are removed from  $G$  and the protocol repeats.

Each iteration of this protocol removes at least half the nodes so that there are at most  $g$  iterations. The communication per iteration is at most  $g + 1$  (to either send one of  $2^g$  nodes or that no good node exists).

## B Proof of Lemma 5.4

**Lemma B.1.**  $\mathcal{F}$  is a correct OR-FBDD with no-op nodes.

*Proof.* We need to show that  $\mathcal{F}$  is acyclic and that every path reads a variable at most once. These two properties follow from the lemma:

**Lemma B.2.** If  $u$  is a leaf node in  $\mathcal{D}$  labeled by the variable  $X$  and there exists a non-trivial path (with at least one edge) between the nodes  $(u, s), (v, s')$  in  $\mathcal{F}$ , then the variable  $X$  does not occur in  $\mathcal{D}_v$ .

This lemma implies that  $\mathcal{F}$  is acyclic: a cycle in  $\mathcal{F}$  implies a non-trivial path from some node  $(u, s)$  to itself, but  $X \in \mathcal{D}_u$ . It also implies that every path in  $\mathcal{F}$  is read-once: if a path tests a variable  $X$  twice, first at  $(u, s)$  and again at  $(u_1, s_1)$ , then  $X \in \mathcal{D}_{u_1}$  contradicting the claim.

To prove the lemma, suppose to the contrary that there exists an OR-FBDD node  $(u, s)$  such that  $u$  is a leaf labeled with  $X$  and that there exists a path from  $(u, s)$  to  $(v, s')$  in  $\mathcal{F}$  such that  $X$  occurs in  $\mathcal{D}_v$ . Choose  $v$  such that  $\mathcal{D}$  is maximal; i.e. there is no path from  $(u, s)$  to some  $(v', s'')$  such that  $\mathcal{D}_v \subset \mathcal{D}_{v'}$  and  $X$  occurs in  $\mathcal{D}_{v'}$ . Consider the last edge on the path from  $(u, s)$  to  $(v, s')$  in  $\mathcal{F}$ :

$$(u, s), \dots, (w, s''), (v, s').$$

Observe that  $(w, v)$  is not an edge in  $\mathcal{D}$  since  $\mathcal{D}_v$  is maximal, and  $(u, v)$  is not an edge in  $\mathcal{D}$  since  $u$  was a leaf. Therefore the edge from  $(w, s'')$  to  $(v, s')$  is Type 3. So  $\mathcal{D}$  has an AND-node  $z$  with children  $v_l, v$  and the last path edge is of the form  $(w, s' \cup \{e\}), (v, s')$  where  $e = (z, v_l)$  is the light edge of  $z$ . We claim that  $e \notin s$ , so that it is not present at the beginning of the path. If  $e \in s$  then, since  $s \in S(u)$ , we have  $u$ , which queries  $X$ , in  $\mathcal{D}_{v_l}$ . Together with the assumption that some node in  $\mathcal{D}_v$  queries  $X$ , we see that descendants of the two children  $v_l, v$  of AND-node  $z$  query the same variable which contradicts that  $\mathcal{D}$  is a DNNF. On the other hand,  $e \in s''$ . Now, the first node on the path where  $e$  was introduced must have an edge of the form  $(z, s_1), (v_l, s_1 \cup \{e\})$ . But now we have a path from  $(u, s)$  to  $(z, s_1)$  with  $X \in \mathcal{D}_z \supset \mathcal{D}_v$ , contradicting the maximality of  $v$ .  $\square$

The next proposition says that on accepting paths  $P = \{(u_1, s_1), (u_2, s_2), \dots (u_\ell, s_\ell)\}$  in the constructed OR-FBDD  $\mathcal{F}$ , the sequence of sets  $(s_1, \dots, s_\ell)$  behaves like the sequence of states of a stack. We will use this characterization of paths in the proof of Lemma B.4.

**Proposition B.3.** *Suppose that  $\mathcal{F}$  has been constructed from a DNNF  $\mathcal{D}$  and that  $P = \{(u_1, s_1), (u_2, s_2), \dots (u_\ell, s_\ell)\}$  is a path in  $\mathcal{F}$  consistent with a variable assignment  $\theta$ . If, for  $j < i$ , we have  $e_1 \in s_j$ ,  $e_1 \in s_i$ , and  $e_2 \in s_i \setminus s_j$ , then for no  $k > i$  do we have both  $e_2 \in s_k$  and  $e_1 \notin s_k$ .*

*Proof.* Suppose the statement is false. Then there must exist a Type 3 edge  $((w, s \cup \{e_1, e_2\}), (v_r, s \cup \{e_2\}))$  in the path  $P$ , where  $w \in v_l$ . However, we cannot have  $e_2 \in S(v_r)$ :  $e_2$  was an edge in  $\mathcal{D}_{v_l}$  because  $(w, s \cup \{e_1, e_2\})$  was reachable in  $\mathcal{F}$  meaning that  $e_2 \in S(w)$ .  $\square$

**Lemma B.4.**  $\mathcal{F}$  computes the same function as  $\mathcal{D}$ . That is,  $\Phi_{\mathcal{F}}[\theta] = \Phi_{\mathcal{D}}[\theta]$  for all variable assignments  $\theta$ .

*Proof.* Suppose that  $\Phi_{\mathcal{F}}[\theta] = 1$ . Then there exists a path  $P$  in  $\mathcal{F}$  consistent with  $\theta$  that ends in a 1-sink node.

If  $P = \{(u_1, s_1), (u_2, s_2), \dots (u_\ell, s_\ell)\}$  has no Type 1 edges then it also has no Type 3 edges. Therefore  $(u_1, \dots, u_\ell)$  is a path of neutral edges in  $\mathcal{D}$  consistent with  $\theta$  to a 1-sink with no AND-nodes along the way, thus  $\Phi_{\mathcal{D}}[\theta] = 1$ .

Otherwise, let

$$S = \{(u_i, s_i), (u_{i+1}, s_{i+1}), \dots, (u_{i+j}, s_{i+j})\}$$

be a sub-path of  $P$ . We say that  $S$  corresponds to an accepting sub-DAG rooted at  $u$  if there is a sub-DAG of  $\mathcal{D}$  rooted at the node  $u$  whose OR nodes have fanout 1, AND nodes have full fanout, leaves are all 1-sinks under  $\theta$ , and whose edges are  $u_i, u_{i+1}, \dots, u_{i+j}$ .

Starting from an empty path in  $\mathcal{F}$ , we will work backwards from the end of  $P$ , adding two possible kinds of sub-path: with all Type 2 edges removed, the first contains exactly one Type 1 edge followed by one or more Type 3 edges. The second kind, with all Type 2 edges removed, contains only one Type 1 edge and no Type 3 edges. It is possible to construct  $P$  using these two types of subpath by Proposition B.3, which says that

$$s_1, s_2 \dots s_\ell$$

is the sequence of states of a stack where, as we traverse the path  $P$ , its Type 1 edges push light edges while its Type 3 edges pop them. As  $P$  is an accepting path, we also have that  $s_\ell = \emptyset$ . We will show that both

of these types of additions give a path corresponding to an accepting sub-DAG rooted at all AND nodes mentioned in the Type 1 edges of the sub-path.

In the first case, say we add the sub-path  $S_h$  to the tail path  $S_t$  to form  $S = S_h S_t$ . Suppose  $S_h$  contains the Type 1 edge  $((u_h, s), (v_l, s \cup \{e\}))$ , and that  $S_t$  corresponds to an accepting sub-DAG respecting the first AND node in  $S_t$ , which we call  $u_t$ . We wish to show that  $S$  corresponds to an accepting sub-DAG rooted at  $u_h$ .  $S_h$  must contain a Type 3 edge popping  $e$ , hence there is a path in  $\mathcal{D}$  from  $v_l$  to a 1-sink that is consistent with  $\theta$ . Therefore  $S_h$  corresponds to an accepting sub-DAG rooted at  $v_l$  (there are no AND-nodes along the way so the sub-DAG is the path). Further, since we can find a path of neutral edges in  $\mathcal{D}$  from the sibling node of  $v_l$ ,  $v_r$ , to  $u_t$  (these come from the portion of  $P$  between the Type 3 edge popping  $e$  and the Type 1 edge  $((u_h, s), (v_l, s \cup \{e\}))$ ),  $S$  corresponds to an accepting sub-DAG rooted at  $v_r$ . Therefore,  $S$  corresponds to an accepting sub-DAG rooted at  $u_h$ .

In the second case, we add a Type 1 edge  $((u_h, s), (v_l, s \cup \{e\}))$  to the tail path  $S_t$ , which corresponds to an accepting sub-DAG rooted at  $u_t$ , the first AND node in  $S_t$ . Then  $u_t$  must appear in  $\mathcal{D}_{v_l}$ . Otherwise the first Type 1 edge in  $S_t$  comes after we pop  $e$ , but then  $S_t$  began with the Type 3 edge popping  $e$ . This cannot happen because of our inductive assumption that we add sub-paths that begin with a Type 1 edge. So  $S_t$  gives a path of neutral edges in  $\mathcal{D}$  from  $v_l$  to  $u_t$  (this is the sub-path in between the added Type 1 edge and the first Type 1 edge in  $S_t$ ). Since  $S_t$  corresponds to an accepting sub-DAG rooted at  $u_t$ , it also corresponds to an accepting sub-DAG rooted at  $v_l$ . Similarly,  $S_t$  gives a neutral edge path in  $\mathcal{D}$  from  $v_r$  to the first AND node mentioned after popping  $e$ . Again, from our inductive hypothesis,  $S_t$  thus corresponds to an accepting sub-DAG rooted at  $v_r$ . Therefore  $S$  corresponds to an accepting sub-DAG rooted at  $u_h$ .

Now suppose  $\Phi_{\mathcal{D}}[\theta] = 1$ . Then  $\mathcal{D}$  has an accepting sub-DAG  $\mathcal{D}_{\theta}$  respecting  $\theta$ . We can find an accepting path in  $\mathcal{F}$  from edges coming from  $\mathcal{D}_{\theta}$ . This path will follow a left-to-right traversal of  $\mathcal{D}_{\theta}$ , keeping track of light edges pushed and popped. The Type 1 and Type 2 edge portions of this traversal (moving left down the tree) directly translate to the appropriate edges in  $\mathcal{F}$ . The necessary Type 3 edges for this traversal also exist in  $\mathcal{F}$  since  $\mathcal{D}_{\theta}$  only has 1-sinks. At the end of the traversal we will have popped all light edges and be at a 1-sink in  $\mathcal{D}_{\theta}$  so we will be at a 1-sink for  $\mathcal{F}$ .  $\square$